KARTA OPISU MODUŁU KSZTAŁCENIA

Nazwa modułu/przedmiotu	Kod
Inżynieria oprogramowania	**1010511351010510082**

Kierunek studiów	Profil kształcenia (ogólnoakademicki, praktyczny)	Rok / Semestr
Informatyka	**ogólnoakademicki**	**3 / 5**
Ścieżka obieralności/specjalność	Przedmiot oferowany w języku:	Kurs (obligatoryjny/obieralny)
-	**polski**	**obligatoryjny**

Stopień studiów:	**Forma studiów** (stacjonarna/niestacjonarna)
I stopień	**stacjonarna**

Godziny	Liczba punktów
Wykłady: **30** Ćwiczenia: **-** Laboratoria: **30** Projekty/seminaria: **-**	**3**

Status przedmiotu w programie studiów (podstawowy, kierunkowy, inny)	(ogólnouczelniany, z innego kierunku)
kierunkowy	**z danego kierunku**

Obszar(y) kształcenia i dziedzina(y) nauki i sztuki	**Podział ECTS (liczba i %)**
nauki techniczne	**3 100%**

Odpowiedzialny za przedmiot / wykładowca:

dr inż. Miroslaw Ochodek
email: Miroslaw.Ochodek@cs.put.poznan.pl
tel. 61 665 2944
Wydział Informatyki
ul. Piotrowo 3, 60-965 Poznań

Odpowiedzialny za przedmiot / wykładowca:

dr inż. Wojciech Complak
email: wojciech.complak@cs.put.poznan.pl
tel. 61 665 2983
Wydział Informatyki
ul. Piotrowo 3, 60-965 Poznań

Wymagania wstępne w zakresie wiedzy, umiejętności, kompetencji społecznych:

1	**Wiedza:**	Student rozpoczynający ten przedmiot powinien posiadać podstawową wiedzę z podstaw programowania, narzędzi informatyki, algorytmów i struktur danych, programowania obiektowego, architektury systemów komputerowych, systemów baz danych.
2	**Umiejętności:**	Student powinien posiadać umiejętność rozwiązywania podstawowych problemów z zakresu programowania oraz umiejętność pozyskiwania informacji ze wskazanych źródeł.
3	**Kompetencje społeczne**	Student powinien rozumieć konieczność poszerzania swoich kompetencji / mieć gotowość do podjęcia współpracy w ramach zespołu. Ponadto w zakresie kompetencji społecznych student musi prezentować takie postawy jak uczciwość, odpowiedzialność, wytrwałość, ciekawość poznawcza, kreatywność, kultura osobista, szacunek dla innych ludzi.

Cel przedmiotu:

-Przekazanie studentom podstawowej wiedzy z inżynierii oprogramowania, w zakresie organizacji przebiegu przedsięwzięcia programistycznego, określania wymagań, modelowania systemów, projektowania oprogramowania, zapewniania jakości (w tym testowania oprogramowania), narzędzi wspomagających wytwarzanie oprogramowania

(w tym narzędzi zarządzania wersjami).

- Rozwijanie u studentów umiejętności rozwiązywania prostych problemów z zakresu projektowania, budowy i testowania oprogramowania, wykorzystania narzędzi wspomagających wytwarzanie oprogramowania, modyfikowania u wykorzystania komponentów programistycznych.

- Kształtowanie u studentów umiejętności efektywnej pracy jako analityk/projektant/programista w zespole programistycznym pracującym zgodnie z klasycznymi lub zwinnymi (Agile) metodykami.

Efekty kształcenia i odniesienie do kierunkowych efektów kształcenia

Wiedza:

1. ma uporządkowaną i podbudowaną teoretycznie wiedzę ogólną w zakresie kluczowych zagadnień inżynierii oprogramowania, a w szczególności zna obszary, którymi zajmuje się inżynieria oprogramowania oraz historię i genezę jej powstania - [K1st_W4]

2. ma wiedzę o istotnych kierunkach rozwoju i najważniejszych osiągnięciach w dziedzinie inżynierii oprogramowania w obszarach inżynierii wymagań, modelowania oprogramowania, zarządzania konfiguracją, weryfikacji oprogramowania, oraz zarządzania projektami informatycznymi - [K1st_W5]

3. ma podstawową wiedzę o cyklu życia projektu informatycznego - [K1st_W6]

4. zna metody specyfikowania wymagań w projektach informatycznych (przypadki użycia, wymagania pozafunkcjonalne) - [K1st_W7]

5. zna podstawowe założenia metodyk Scrum oraz PRINCE2 - [K1st_W7]

6. zna przynajmniej jedną metodę szacowania pracochłonności wytwarzenia oprogramowania - [K1st_W7]

7. zna przynajmniej jedno narzędzie zarządzania wersjami oprogramowania (Git, SVN) - [K1st_W7]

8. zna przynajmniej jedno narzędzie służące do budowania oprogramowania (Ant, Maven) - [K1st_W7]

9. zna wybrane elementy notacji UML (diagram klas, diagram maszyny stanów, diagram sekwencji) - [K1st_W7]

10. zna metody i narzędzia testowania jednostkowego oprogramowania (xUnit) - [K1st_W7]

11. zna wybrane metody i narzędzia testowania akceptacyjnego (testy eksploracyjne, automatyzacja testów, akceptacyjne testy wydajnościowe) - [K1st_W7]

12. zna wybrane metryki rozmiaru i jakości kodu obiektowego (złożoność cyklomatyczna, brak spójności, miary sprzężenia) oraz narzędzia do monitorowania miar w projekcie (SonarQube) - [K1st_W7]

Umiejętności:

1. potrafi tworzyć dokumentację kodu (na przykładzie języka Java) - [K1st_U2]

2. potrafi korzystać z narzędzi zarządzania zadaniami i śledzeniem postępu pracy grupowej w projekcie informatycznym (na przykładzie narzędzia Redmine) - [K1st_U2]

3. potrafi uczestniczyć w spotkaniach projektowych w metodyce Scrum w roli członka zespołu deweloperskiego (planowanie, przegląd i retrospektywa sprintu) - [K1st_U2]

4. potrafi formułować wizję produktu informatycznego uwzględniając potrzeby grupy docelowej - [K1st_U5]

5. potrafi identyfikować i oceniać ryzyka w projekcie informatycznym - [K1st_U6]

6. potrafi ocenić architekturę oprogramowania oceniając punkty wrażliwości i kompromisu - [K1st_U9]

7. potrafi projektować i przeprowadzić testy akceptacyjne oprogramowania - [K1st_U9]

8. potraf projektować i wykonywać testy jednostkowe oprogramowania - [K1st_U9]

9. potrafi tworzyć modele obiektowe w notacji UML (model klas, maszyny stanowej, sekwencji) - [K1st_U10]

10. potrafi specyfikować wymagania funkcjonalne w postaci przypadków użycia - [K1st_U10]

11. potrafi specyfikować wymagania pozafunkcjonalne - [K1st_U10]

12. potrafi zaprojektować interfejs systemu informatycznego spełniający zadane wymagania - [K1st_U14]

13. potrafi współpracować w zespole deweloperskim realizującym projekt informatyczny według wytycznych metodyki Scrum - [K1st_U18]

14. potrafi zaplanować prace zespołu programistycznego zgodnie z zaleceniami metodyki Scrum - [K1st_U18]

Kompetencje społeczne:

1. ma świadomość tego, że narzędzia oraz biblioteki programistyczne podlegają ciągłym i częstym zmianom (np. na podstawie zmian w bibliotece JUnit, czy narzędzi do zarządzania wersjami) - [K1st_K1]

2. zna przykłady i rozumie przyczyny wadliwie działających systemów informatycznych, które doprowadziły do poważnych strat finansowych, społecznych lub też do poważnej utraty zdrowia, a nawet życia - [K1st_K2]

3. potrafi identyfikować rzeczywiste problemy o znaczeniu komercyjnym, które można rozwiązać poprzez implementacje i wdrożenie systemów informatycznych - [K1st_K3]

Sposoby sprawdzenia efektów kształcenia

Efekty kształcenia przedstawione wyżej weryfikowane są w następujący sposób:

Ocena formująca:

a) w zakresie wykładów: na podstawie odpowiedzi na pytania oraz udział w quizach w trakcie zajęć wykładowych

b) w zakresie laboratorium: na podstawie oceny bieżącego postępu realizacji zadań w trakcie laboratorium

Ocena podsumowująca:

W zakresie oceny efektów kształcenia dotyczących nabytych umiejętności oraz kompetencji społecznych (głównie ocena z zajęć laboratoryjnych):

a) w zależności od stopnia realizacji zadań na poszczególnych zajęciach laboratoryjnych student może otrzymać od 0 do 15 punktów (0 ? nieobecność lub zupełny brak zaangażowania, 10 ? spełnione minimum z perspektywy założonych efektów kształcenia, 15 ? pełna realizacja materiału w odniesieniu do założonych efektów kształcenia). Student nieobecny na zajęciach może odrobić zajęcia w innym terminie lub za zgodą prowadzącego wykonać zadania w domu. Każdy student może uzyskać łącznie od 0 do 165 punktów.

b) w trakcie semestru studenci realizują projekt grupowy (3-5 osób) według zaleceń metodyki Scrum. Projekt składa się z dwóch sprintów (iteracji). W każdym sprincie zespół może uzyskać od 0-n*100 (gdzie n jest liczbą osób w zespole) punktów w zależności od stopnia realizacji zadań. Każdy z członków zespołu może otrzymać maksymalnie 100 punktów za sprint co daje łącznie maksymalnie 200 punktów.

Na podstawie sumy uzyskanych punktów wyznaczona zostaje ocena końcowa według następujący progów:

- >=310 - 5.0
- <270, 310) - 4.5
- <230, 270) - 4.0
- <190, 230) - 3.5
- <150, 190) - 3.0
- mniej niż 150 - 2.0

W zakresie efektów kształcenia dotyczących nabytej wiedzy:

a) w trakcie wykładów studenci rozwiązują quizy oraz krótkie zadania o charakterze problemowym. Za dostarczenie akceptowalnego rozwiązania student otrzymuje 1%.

b) test wyboru obejmujący 22 pytania jednokrotnego wyboru (jedna prawidłowa odpowiedź) oraz 4 pytania z możliwie jedną lub wieloma poprawnymi odpowiedziami (typ pytania jest jawnie wskazany w teście). Za udzielenie poprawnej odpowiedzi na pytanie student otrzymuje 1 punkt. Punkty przeliczane są na skalę procentową.

Na podstawie uzyskanych punktów procentowych wyznaczana jest ocena końcowa według skali:

- >= 90% - 5.0
- <80%, 90%) ? 4.5
- <70%, 80%) ? 4.0
- <60%, 70%) ? 3.5
- <50%, 60%) ? 3.0
- mniej niż 50% - 2.0

Studenci mogą monitorować otrzymywane punkty za pomocą portalu http://io.cs.put.poznan.pl.

Treści programowe

Program przedmiotu obejmuje następujące zagadnienia:

- Wprowadzenie w tym znaczenie i rola wytwarzania oprogramowania we współczesnym świecie, wizja projektu informatycznego, konsekwencje błędów w oprogramowaniu, zakres tematyczny inżynierii oprogramowania
- Zarządzanie konfiguracją oprogramowania (w tym systemy zarządzania wersjami, narzędzia automatycznego budowania oprogramowania oraz praktyki ciągłej integracji)
- Wymagania funkcjonalne (w tym przypadki użycia)
- Wymagania pozafunkcjonalne (w tym norma ISO 25010)
- Modelowanie i analiza oprogramowania (w tym notacja UML)
- Projektowanie oprogramowania (w tym wzorce projektowe)
- Architektura oprogramowania
- Metodyki zarządzania projektami (w tym zwinne metodyki zarządzania projektami ? Scrum, XP)
- Zarządzanie jakością oprogramowania (m.in. pomiar w procesie wytwarzania oprogramowania)
- Testowanie oprogramowania (jednostkowe, integracyjne, akceptacyjne, pozafunkcjonalne)

Program zajęć laboratoryjnych obejmuje następujące zagadnienia:

- Zintegrowane środowiska programistyczne, np. InteliJ
- Standardy kodowania oraz dokumentowanie oprogramowania, np. javadoc
- Ocena ryzyka w projektach informatycznych
- Narzędzia zarządzania konfiguracją oprogramowania, np. Git, Ant, Maven
- Dokumentowanie wymagań funkcjonalnych z wykorzystaniem metody przypadków użycia
- Dokumentowanie wymagań pozafunkcjonalnych
- Ocena ryzyka w projekcie informatycznym
- Modelowanie systemów w notacji UML
- Projektowanie oprogramowania z wykorzystaniem wzorców projektowych
- Testowanie oprogramowania, w tym testy jednostkowe i akceptacyjne
- Praktyczna realizacja mini-projektu według zaleceń metodyki Scrum

Metody dydaktyczne:

Mini-projekt realizowany jest według autorskiej metody opisanej w poniższym artykule naukowym:

Ochodek, Mirosław. "A Scrum-Centric Framework for Organizing Software Engineering Academic Courses." In Towards a Synergistic Combination of Research and Practice in Software Engineering, pp. 207-220. Springer, Cham, 2018.

Pozostałe metody dydaktyczne obejmują:

a) wykład: prezentacja multimedialna, prezentacja ilustrowana przykładami podawanymi na tablicy, rozwiązywanie zadań, studium przypadków.

b) ćwiczenia laboratoryjne: rozwiązywanie zadań, ćwiczenia praktyczne, dyskusja, praca w zespole, pokaz multimedialny, warsztaty, demonstracja.

Literatura podstawowa:

1. A. Jaszkiewicz, Inżynieria oprogramowania, Helion, 1997.

2. K. Schwaber, J. Sutherland, The Scrum Guide: Przewodnik po Scrumie: Reguły Gry, http://www.scrumguides.org, (dostępny online), 2017

Literatura uzupełniająca:

1. Wzorce projektowe w języku Java: https://www.tutorialspoint.com/design_pattern

2. Ochodek, Miroslaw, J. Nawrocki, and K. Kwarciak. "Simplifying effort estimation based on Use Case Points." Information and Software Technology 53.3 (2011): 200-213.

3. Nawrocki, Jerzy R., Miroslaw Ochodek, Jakub Jurkiewicz, Sylwia Kopczynska, and Bartosz Alchimowicz. "Agile Requirements Engineering: A Research Perspective." In SOFSEM, pp. 40-51. 2014.

Bilans nakładu pracy przeciętnego studenta

Czynność	Czas (godz.)

1. Udział w zajęciach laboratoryjnych	30
2. Zadania domowe / realizacja mini-projektu (w ramach pracy własnej)	8
3. Udział w konsultacjach związanych z realizacją procesu kształcenia, w szczególności ćwiczeń laboratoryjnych / projektu	1
4. Udział w wykładach	30
5. Zapoznanie się ze wskazaną literaturą / materiałami dydaktycznymi	3
6. Przygotowanie do testu zaliczeniowego	3

Obciążenie pracą studenta

forma aktywności	godzin	ECTS
Łączny nakład pracy	75	3
Zajęcia wymagające bezpośredniego kontaktu z nauczycielem	61	2
Zajęcia o charakterze praktycznym	39	2